Assignment 4

Rashik Mahmud Orchi-B00968298

16/06/2024

```
rm(list=ls())
```

There are 3-4 packages you will need to install for today's practical: `install.packages(c("xgboost", "eegkit", "forecast", "tseries", "caret"))` apart from that everything else should already be available on your system.

If you are using a newer Mac you may have to also install quartz to have everything work (do this if you see errors about `X11` during install/execution).

I will endeavour to use explicit imports to make it clear where functions are coming from (functions without `library_name::` are part of base R or a function we've defined in this notebook).

EEG Eye Detection Data

One of the most common types of medical sensor data (and one that we talked about during the lecture) are Electroencephalograms (EEGs).
These measure mesoscale electrical signals (measured in microvolts) within the brain, which are indicative of a region of neuronal activity. Typically, EEGs involve an array of sensors (aka channels) placed on the scalp with a high degree of covariance between sensors.

As EEG data can be very large and unwieldy, we are going to use a relatively small/simple dataset today from this paper.

This dataset is a 117 second continuous EEG measurement collected from a single person with a device called a "Emotiv EEG Neuroheadset". In combination with the EEG data collection, a camera was used to record whether person being recorded had their eyes open or closed. This was eye status was then manually annotated onto the EEG data with `1` indicated the eyes being closed and `0` the eyes being open. Measures microvoltages are listed in chronological order with the first measured value at the top of the dataframe.

Let's parse the data directly from the `h2o` library's (which we aren't actually using directly) test data S3 bucket:

```
eeg_url <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/eeg/eeg_eyestate_splits.csv"
eeg_data <- read.csv(eeg_url)

# add timestamp
Fs <- 117 / nrow(eeg_data)
eeg_data <- transform(eeg_data, ds = seq(0, 116.99999, by = Fs), eyeDetection = as.factor(eyeDetection))
print(table(eeg_data$eyeDetection))
```

```
##
##    0    1
## 8257 6723
```

```r
# split dataset into train, validate, test
eeg_train <- subset(eeg_data, split == 'train', select = -split)
print(table(eeg_train$eyeDetection))
```

```
##
##    0    1
## 4916 4072
```

```r
eeg_validate <- subset(eeg_data, split == 'valid', select = -split)
eeg_test <- subset(eeg_data, split == 'test', select = -split)
```

0 Knowing the `eeg_data` contains 117 seconds of data, inspect the `eeg_data` dataframe and the code above to and determine how many samples per second were taken?

```r
sampling_rate <- nrow(eeg_data) / 117
print(sampling_rate)
```

```
## [1] 128.0342
```

1 How many EEG electrodes/sensors were used?

Total 14 EEG electrodes/sensors were used.

```r
print(colnames(eeg_data))
```

```
##  [1] "AF3"          "F7"           "F3"           "FC5"          "T7"
##  [6] "P7"           "O1"           "O2"           "P8"           "T8"
## [11] "FC6"          "F4"           "F8"           "AF4"          "eyeDetection"
## [16] "split"        "ds"
```

```r
print(ncol(eeg_data))
```

```
## [1] 17
```

Exploratory Data Analysis

Now that we have the dataset and some basic parameters let's begin with the ever important/relevant exploratory data analysis.

First we should check there is no missing data!

```r
sum(is.na(eeg_data))
```

```
## [1] 0
```

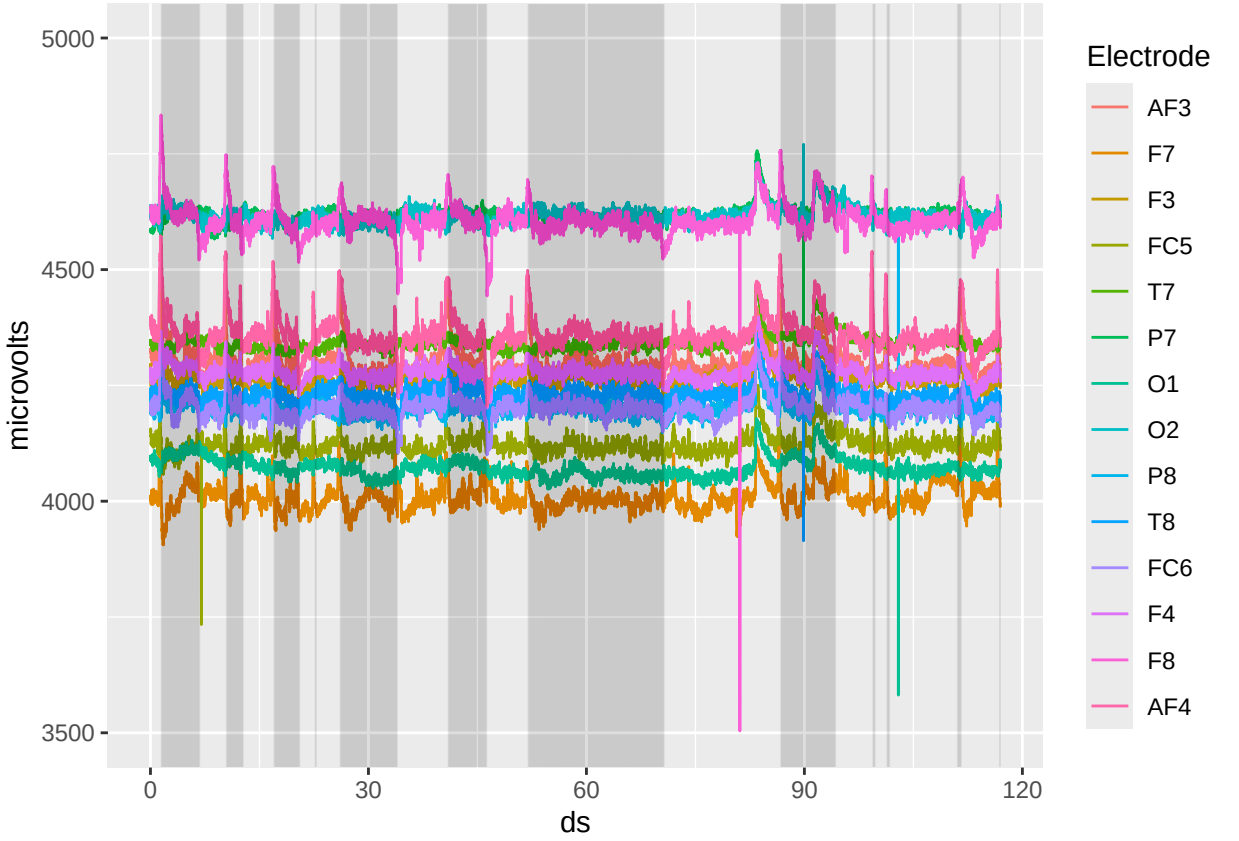
Great, now we can start generating some plots to look at this data within the time-domain.

First we use `reshape2::melt()` to transform the `eeg_data` dataset from a wide format to a long format expected by `ggplot2`.

Specifically, this converts from "wide" where each electrode has its own column, to a "long" format, where each observation has its own row. This format is often more convenient for data analysis and visualization, especially when dealing with repeated measurements or time-series data.

We then use `ggplot2` to create a line plot of electrode intensities per sampling time, with the lines coloured by electrode, and the eye status annotated using dark grey blocks.

```
melt <- reshape2::melt(eeg_data %>% dplyr::select(-split), id.vars=c("eyeDetection", "ds"), variable.nam


ggplot2::ggplot(melt, ggplot2::aes(x=ds, y=microvolts, color=Electrode)) +
  ggplot2::geom_line() +
  ggplot2::ylim(3500,5000) +
  ggplot2::geom_vline(ggplot2::aes(xintercept=ds), data=dplyr::filter(melt, eyeDetection==1), alpha=0.0
```



2 Do you see any obvious patterns between eyes being open (dark grey blocks in the plot) and the EEG intensities? During the periods when the eyes are presumed to be open (indicated by the dark gray blocks), the EEG signal amplitudes across multiple electrode channels initially spike or increase substantially. However, just before the presumed eye-closing event, there is a noticeable decrease or dampening of the signal amplitudes. So during the transition phase of opeing and closign of eye we see more chnages in the value of microvolts across the electrodes. The pattern seems to be:

1. Eyes open - EEG signal amplitudes spike/increase substantially across channels.

2. Sustained open eye period - EEG amplitudes remain elevated but stabilize.

3. Just before closing eyes - EEG amplitudes decrease or dampen.

4. Eyes closed - EEG amplitudes return to a lower baseline level.

This pattern of amplitudes peaking right after opening the eyes, remaining elevated during the open eye period, and then decreasing slightly just before closing the eyes could be related to the specific neurophysiological processes involved in visual attention, processing of visual stimuli, and the preparatory processes leading up to eye closure.

3 Similarly, based on the distribution of eye open/close state over time to anticipate any temporal correlation between these states?

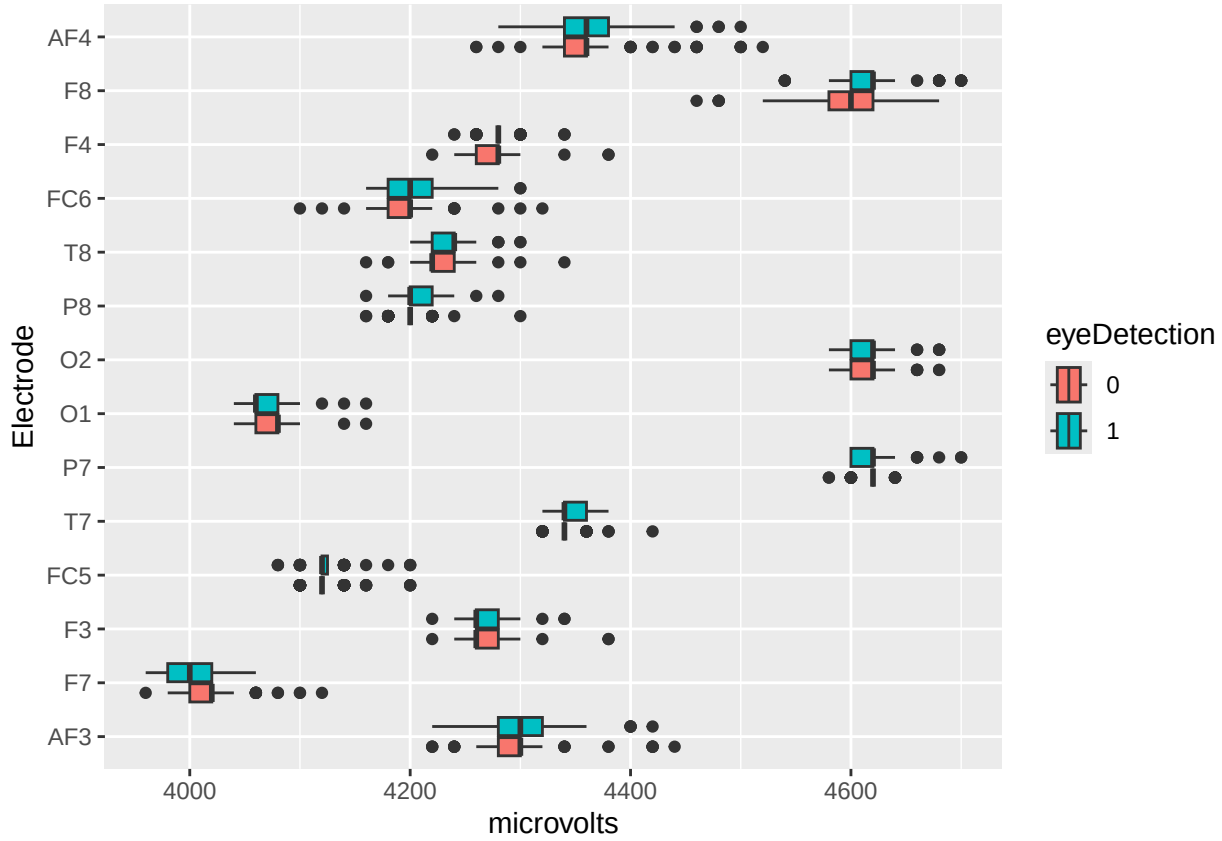
Temporal Correlation these states are discussed below:

- **Duration Variability**:
  - The durations of the eye open and eye close periods vary.
  - Some open periods are relatively short, while others are more prolonged, and the same is true for the closed periods.
- **Clustering**:
  - Instances where several consecutive open or closed periods occur in clusters are observed.
  - This suggests some temporal dependence or correlation within shorter time windows.
- **Transition Dynamics**:
  - Transitions between open and closed states appear relatively abrupt.
  - Signal amplitudes change rapidly during the transition periods.
- **Consistency in Patterns**:
  - While the exact durations vary, there is a noticeable pattern where eye states change at somewhat regular intervals.
  - This indicates a structured alternation between eyes being open and closed.
- **Correlation with EEG Intensities**:
  - Visual inspection suggests certain EEG intensity patterns might correspond with the eye open states.
  - Some electrodes show spikes in intensity around the transitions.
- **Anticipation of Temporal Correlation**:
  - Given the observed structured alternation, it can be anticipated that the eye open and close states are temporally correlated.
  - This regularity can be used to predict future states, assuming the pattern continues consistently throughout the data.

Let's see if we can directly look at the distribution of EEG intensities and see how they related to eye status.

As there are a few extreme outliers in voltage we will use the `dplyr::filter` function to remove values outwith of 3750 to 50003. The function uses the `%in%` operator to check if each value of microvolts is within that range. The function also uses the `dplyr::mutate()` to change the type of the variable eyeDetection from numeric to a factor (R's categorical variable type).

```
melt_train <- reshape2::melt(eeg_train, id.vars=c("eyeDetection", "ds"), variable.name = "Electrode", va

# filter huge outliers in voltage
filt_melt_train <- dplyr::filter(melt_train, microvolts %in% (3750:5000)) %>% dplyr::mutate(eyeDetectio

ggplot2::ggplot(filt_melt_train, ggplot2::aes(y=Electrode, x=microvolts, fill=eyeDetection)) + ggplot2:
```



Plots are great but sometimes so it is also useful to directly look at the summary statistics and how they related to eye status. We will do this by grouping the data based on eye status and electrode before calculating the statistics using the convenient `dplyr::summarise` function.

```
filt_melt_train %>% dplyr::group_by(eyeDetection, Electrode) %>%
    dplyr::summarise(mean = mean(microvolts), median=median(microvolts), sd=sd(microvolts)) %>%
    dplyr::arrange(Electrode)
```

```
## `summarise()` has grouped output by 'eyeDetection'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 28 x 5
## # Groups:   eyeDetection [2]
##    eyeDetection Electrode  mean median    sd
##    <fct>        <fct>     <dbl>  <dbl> <dbl>
##  1 0            AF3       4294.   4300  35.4
##  2 1            AF3       4305.   4300  34.4
##  3 0            F7        4015.   4020  28.4
##  4 1            F7        4007.   4000  24.9
##  5 0            F3        4268.   4260  20.9
##  6 1            F3        4269.   4260  17.4
##  7 0            FC5       4124.   4120  17.3
##  8 1            FC5       4124.   4120  19.2
##  9 0            T7        4341.   4340  13.9
## 10 1            T7        4342.   4340  15.5
## # i 18 more rows
```

4 Based on these analyses are any electrodes consistently more intense or varied when eyes are open?

Analysis of EEG Electrode Activity Based on Eyes Open vs. Eyes Closed

Electrodes with Higher Intensity (Mean and Median Comparison) When Eyes Open

- **Electrode F8:**
  - Mean: 4615.185 (eyes open) vs. 4595.735 (eyes closed)
  - Median: 4620 (eyes open) vs. 4600 (eyes closed)
- **Electrode O2:**
  - Mean: 4619.813 (eyes open) vs. 4615.714 (eyes closed)
  - Median: 4620 (eyes open) vs. 4620 (eyes closed)
- **Electrode F4:**
  - Mean: 4281.319 (eyes open) vs. 4275.085 (eyes closed)
  - Median: 4280 (eyes open) vs. 4280 (eyes closed)
- **Electrode AF4:**
  - Mean: 4369.455 (eyes open) vs. 4363.894 (eyes closed)
  - Median: 4360 (eyes open) vs. 4360 (eyes closed)

Electrodes with Different Variability (Standard Deviation Comparison) When Eyes Open

- **Electrode F7:**
  - Standard Deviation: 24.86399 (eyes open) vs. 28.37562 (eyes closed)
  - Conclusion: Lower variability when eyes are open.
- **Electrode P7:**
  - Standard Deviation: 19.14777 (eyes open) vs. 11.18466 (eyes closed)
  - Conclusion: Higher variability when eyes are open.
- **Electrode O1:**
  - Standard Deviation: 24.28916 (eyes open) vs. 16.93481 (eyes closed)
  - Conclusion: Higher variability when eyes are open.
- **Electrode AF4:**
  - Standard Deviation: 36.01316 (eyes open) vs. 46.58774 (eyes closed)
  - Conclusion: Lower variability when eyes are open.

Time-Related Trends As it looks like there may be a temporal pattern in the data we should investigate how it changes over time.

First we will do a statistical test for stationarity:

```
apply(eeg_train, 2, tseries::adf.test)
```

```
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
```

```
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## $AF3
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.669, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -12.079, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F3
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -11.587, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $FC5
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -11.122, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $T7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.5644, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
```

```
##
## $P7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.7, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $O1
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -7.9495, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $O2
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.3537, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $P8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.69, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $T8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.9902, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $FC6
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -8.6708, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
```

```
##
## $F4
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -10.189, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.642, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $AF4
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.755, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $eyeDetection
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -4.8699, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $ds
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -2.4104, Lag order = 20, p-value = 0.4045
## alternative hypothesis: stationary
```

5 What is stationarity? Stationarity is a fundamental concept in time series analysis. A time series is said to be stationary if its statistical properties such as mean, variance, and covariance are constant over time. This means that the distribution of a stationary process would look the same at different points in time(Chaudhari,2023).

6 Why are we interested in stationarity? What do the results of these tests tell us? (ignoring the lack of multiple comparison correction...)

A time series is considered stationary if its properties do not depend on the time at which the series is observed. Stationarity is an important characteristic because many statistical methods and models assume

or require it.

We're interested in stationarity for time series analysis because it allows us to use simpler and more reliable statistical methods. With stationary data, the statistical properties like mean and variance are constant over time, making it easier to model and predict future values. This is in contrast to non-stationary data, where the properties keep changing, making analysis more complex and unreliable.

The Augmented Dickey-Fuller (ADF) test is used to determine if a time series is stationary or not. In the context of the ADF test:

Null Hypothesis: The time series has a unit root, which means it is non-stationary.
Alternative Hypothesis : The time series does not have a unit root, which means it is stationary.

Interpretation:
1. If the p-value is less than a significance level (commonly 0.05), we reject the null hypothesis in favor of the alternative hypothesis, indicating that the time series is stationary.
2. If the p-value is greater than the significance level, we fail to reject the null hypothesis, indicating that the time series is non-stationary.

The results of Unit root test of all the variables are summarized below:

```
library(knitr)
library("kableExtra")
```

```
##
## Attaching package: 'kableExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

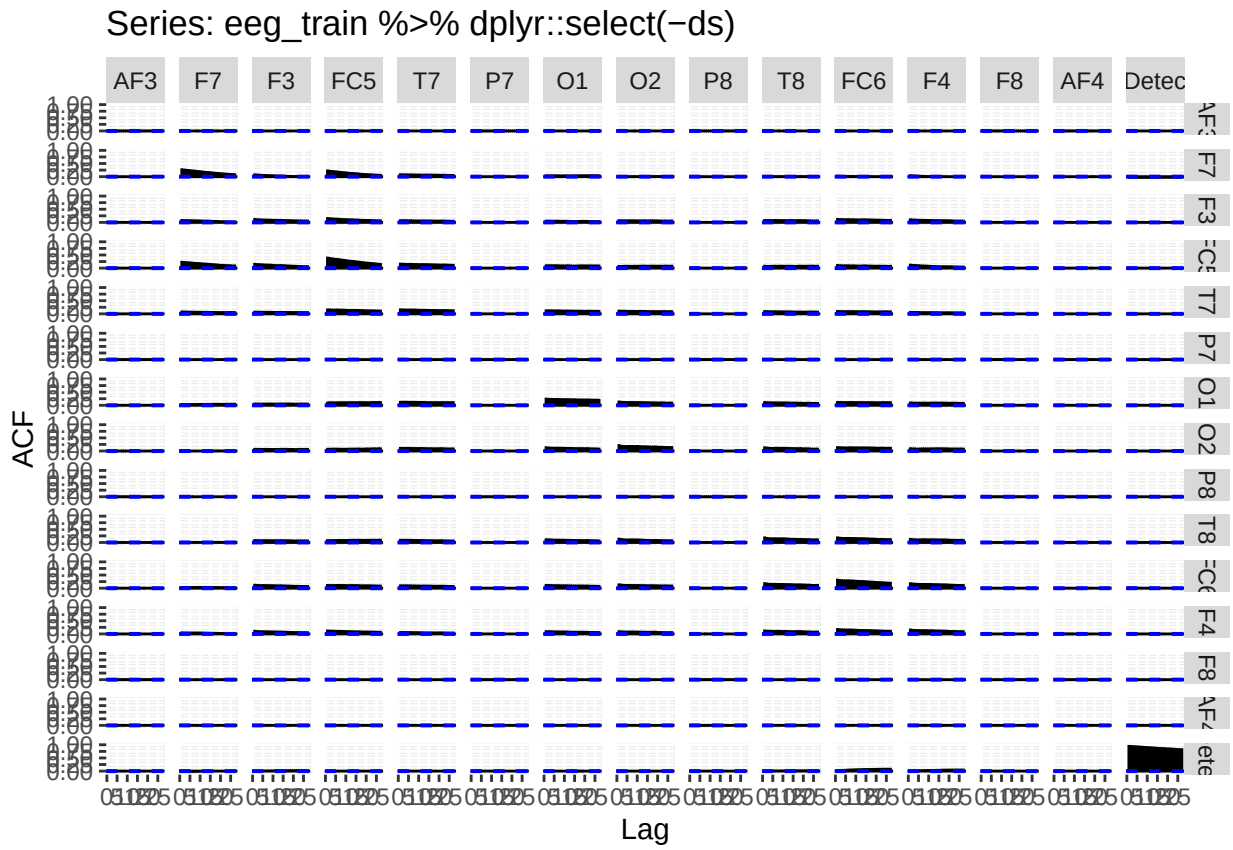
```
data_dictionary <- data.frame()

results <- data.frame(
  Variable = c("AF3", "F7", "F3", "FC5", "T7", "P7", "O1", "O2", "P8", "T8", "FC6", "F4", "F8", "AF4",
  P_Value = c(0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
  Stationarity = c(rep("Stationary", 15), "Non-Stationary")
)


kable(results, col.names = c("Variable", "P_Value", "Stationarity"), align = "l")%>%
  kable_styling(full_width = FALSE, font_size = 11) %>%
  column_spec(3, width = "50em") %>%
  row_spec(1, bold = TRUE) %>%
  row_spec(1:nrow(results), extra_css = "height: 30px;")
```

Variable	P_Value	Stationarity
AF3	**0.0100**	**Stationary**
F7	0.0100	Stationary
F3	0.0100	Stationary
FC5	0.0100	Stationary
T7	0.0100	Stationary
P7	0.0100	Stationary

O1	0.0100	Stationary
O2	0.0100	Stationary
P8	0.0100	Stationary
T8	0.0100	Stationary
FC6	0.0100	Stationary
F4	0.0100	Stationary
F8	0.0100	Stationary
AF4	0.0100	Stationary
eyeDetection	0.0100	Stationary
ds	0.4045	Non-Stationary

The ACF plot displays the cross- and auto-correlation values for different lags (i.e., time delayed versions of each electrode's voltage timeseries) in the dataset. It helps identify any significant correlations between channels and observations at different time points. Positive autocorrelation indicates that the increase in voltage observed in a given time-interval leads to a proportionate increase in the lagged time interval as well. Negative autocorrelation indicates the opposite!

```
forecast::ggAcf(eeg_train %>% dplyr::select(-ds))
```



7 Do any fields show signs of strong autocorrelation (diagonal plots)? Do any pairs of fields show signs of cross-correlation? Provide examples. Fields with Strong Autocorrelation:

Eye Detection: Shows very strong autocorrelation with significant spikes.
FC6: Displays significant autocorrelation patterns.

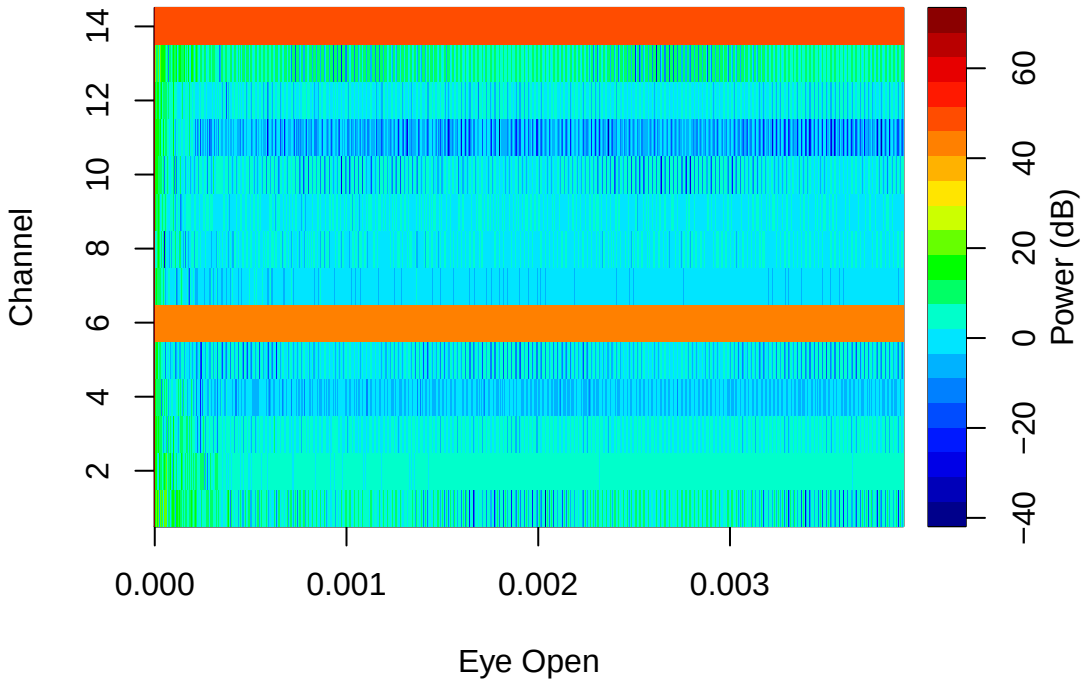
FC5: Indicates strong autocorrelation with noticeable spikes.
T8: Shows clear patterns indicating strong autocorrelation.
F4: Indicates strong autocorrelation with noticeable spikes.
F7: Shows significant spikes and patterns, indicating strong autocorrelation.
O1: Clearly shows autocorrelation patterns.
F8: Indicates strong autocorrelation with noticeable spikes.

Pairs with Noticeable Cross-Correlation:

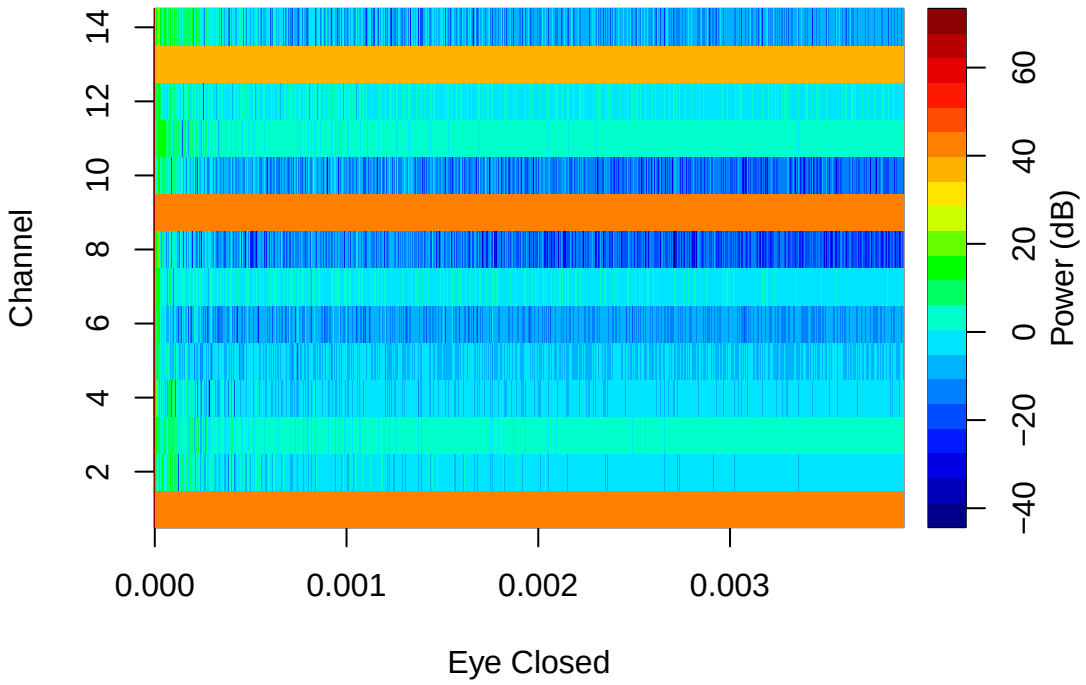
FC6 and FC5: Shows significant cross-correlation, suggesting these fields are related.
FC5 and F3: Displays significant cross-correlation, indicating a relationship between these fields.
FC5 and T7: Shows significant cross-correlation, suggesting these fields are related.
O1 and O2: Displays strong cross-correlation, indicating a relationship between these fields.
O1 and FC5: Shows noticeable cross-correlation, suggesting a connection between these fields.
FC6 and T8: Indicates strong cross-correlation, suggesting these fields are related.

Frequency-Space We can also explore the data in frequency space by using a Fast Fourier Transform. After the FFT we can summarise the distributions of frequencies by their density across the power spectrum. This will let us see if there any obvious patterns related to eye status in the overall frequency distributions.

```
eegkit::eegpsd(eeg_train %>% dplyr::filter(eyeDetection == 0) %>% dplyr::select(-eyeDetection, -ds), Fs
```




```
eegkit::eegpsd(eeg_train %>% dplyr::filter(eyeDetection == 1) %>% dplyr::select(-eyeDetection, -ds), Fs
```



8 Do you see any differences between the power spectral densities for the two eye states? If so, describe them. The PSD plots reveal distinct neural activity patterns based on eye state. When eyes are open, activity concentrates in specific channels linked to visual processing and attention.

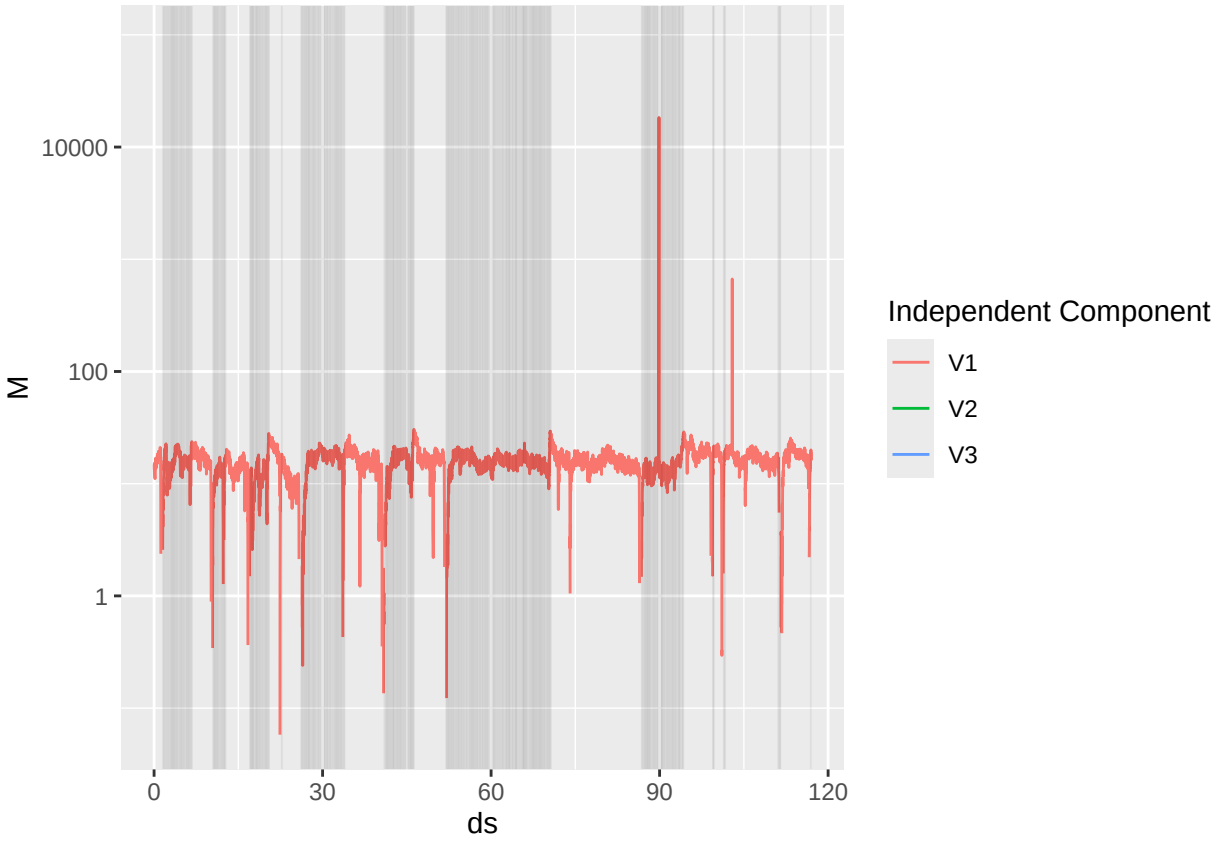
When Eyes are Open:
- Channels 6 and 14 exhibit higher power levels, indicated by the orange and yellow colors.
- The remaining channels mostly display lower power levels, shown by the blue and green colors, suggesting lower neuronal activity or noise in these regions when the eyes are open.

When Eyes are Closed:
-Channels 1, 9, and 13 show higher power levels, indicated by the yellow colors.
-There is a noticeable increase in power across more channels compared to the eyes open state, though there are still significant regions with lower power levels (blue and green colors).

With eyes closed, there is a more widespread increase in activity across multiple channels, indicating the brain transitioning to a relaxed. The differences in PSD plots show that eye state significantly impacts power distribution across EEG channels - eyes open leads to focused activation for visual processing, while eyes closed reflects a diffuse, deactivated neural state characteristic of wakeful rest. These patterns demonstrate how the brain's functional dynamics adapt to changes in eye state between focused visual attention and a globally relaxed neural configuration.

Independent Component Analysis We may also wish to explore whether there are multiple sources of neuronal activity being picked up by the sensors.
This can be achieved using a process known as independent component analysis (ICA) which decorrelates the channels and identifies the primary sources of signal within the decorrelated matrix.

```r
ica <- eegkit::eegica(eeg_train %>% dplyr::select(-eyeDetection, -ds), nc=3, method='fast', type='time')
mix <- dplyr::as_tibble(ica$M)
mix$eyeDetection <- eeg_train$eyeDetection
mix$ds <- eeg_train$ds

mix_melt <- reshape2::melt(mix, id.vars=c("eyeDetection", "ds"), variable.name = "Independent Component")


ggplot2::ggplot(mix_melt, ggplot2::aes(x=ds, y=M, color=`Independent Component`)) +
  ggplot2::geom_line() +
  ggplot2::geom_vline(ggplot2::aes(xintercept=ds), data=dplyr::filter(mix_melt, eyeDetection==1), alpha=
  ggplot2::scale_y_log10()
```



9 Does this suggest eye opening relates to an independent component of activity across the electrodes?

Yes, the plot suggests that eye opening relates to an independent component of activity across the electrodes.

The above plot shows that only V1 (red) is actively displayed along the ds, suggesting that the other components V2 (green) and V3 (blue) are either not present or not visible.

The activity of the V1 component shows significant fluctuations over time. When the eyes are open (dark gray regions), there are frequent and sometimes sharp changes in the magnitude of the V1 component, indicating heightened brain activity. Additionally, at the transition phases of opening and closing the eyes, we can see significant changes in the value of M of the V1 component.

In contrast, during the periods outside the dark gray blocks (presumably when the eyes were closed), the V1 component displays a more stable and consistent activity pattern with lower amplitudes, aligning with the brain's typical response during a relaxed, eyes-closed state.

The distinct patterns observed in V1 during eye opening emphasize the brain's dynamic response to visual stimuli and the act of eye movement, supporting the idea that eye opening relates to an independent component of activity across the electrodes.

Eye Opening Prediction

Now that we've explored the data let's use a simple model to see how well we can predict eye status from the EEGs:

```
# Convert the training and validation datasets to matrices
eeg_train_matrix <- as.matrix(dplyr::select(eeg_train, -eyeDetection, -ds))
eeg_train_labels <- as.numeric(eeg_train$eyeDetection)-1

eeg_validate_matrix <- as.matrix(dplyr::select(eeg_validate, -eyeDetection, -ds))
eeg_validate_labels <- as.numeric(eeg_validate$eyeDetection)-1

# Build the xgboost model
model <- xgboost(data = eeg_train_matrix,
                 label = eeg_train_labels,
                 nrounds = 100,
                 max_depth = 4,
                 eta = 0.1,
                 objective = "binary:logistic")
```

```
## [1]  train-logloss:0.672173
## [2]  train-logloss:0.653965
## [3]  train-logloss:0.638226
## [4]  train-logloss:0.622881
## [5]  train-logloss:0.610129
## [6]  train-logloss:0.599369
## [7]  train-logloss:0.588913
## [8]  train-logloss:0.577916
## [9]  train-logloss:0.568707
## [10] train-logloss:0.560877
## [11] train-logloss:0.553595
## [12] train-logloss:0.546697
## [13] train-logloss:0.540356
## [14] train-logloss:0.535189
## [15] train-logloss:0.528949
## [16] train-logloss:0.523818
## [17] train-logloss:0.517499
## [18] train-logloss:0.513480
## [19] train-logloss:0.509431
## [20] train-logloss:0.504572
## [21] train-logloss:0.501298
## [22] train-logloss:0.499068
## [23] train-logloss:0.493927
## [24] train-logloss:0.488979
## [25] train-logloss:0.485995
## [26] train-logloss:0.484120
## [27] train-logloss:0.480735
## [28] train-logloss:0.476749
## [29] train-logloss:0.475324
```

```
## [30] train-logloss:0.471852
## [31] train-logloss:0.469037
## [32] train-logloss:0.466092
## [33] train-logloss:0.464552
## [34] train-logloss:0.462219
## [35] train-logloss:0.457720
## [36] train-logloss:0.455526
## [37] train-logloss:0.452435
## [38] train-logloss:0.448676
## [39] train-logloss:0.447381
## [40] train-logloss:0.444740
## [41] train-logloss:0.442885
## [42] train-logloss:0.441704
## [43] train-logloss:0.437273
## [44] train-logloss:0.435778
## [45] train-logloss:0.432542
## [46] train-logloss:0.431302
## [47] train-logloss:0.430229
## [48] train-logloss:0.426916
## [49] train-logloss:0.423800
## [50] train-logloss:0.421908
## [51] train-logloss:0.419130
## [52] train-logloss:0.417934
## [53] train-logloss:0.415003
## [54] train-logloss:0.414027
## [55] train-logloss:0.412499
## [56] train-logloss:0.409956
## [57] train-logloss:0.408821
## [58] train-logloss:0.407170
## [59] train-logloss:0.404487
## [60] train-logloss:0.402856
## [61] train-logloss:0.402061
## [62] train-logloss:0.401169
## [63] train-logloss:0.400292
## [64] train-logloss:0.399799
## [65] train-logloss:0.397281
## [66] train-logloss:0.395909
## [67] train-logloss:0.393773
## [68] train-logloss:0.390365
## [69] train-logloss:0.389440
## [70] train-logloss:0.386978
## [71] train-logloss:0.386324
## [72] train-logloss:0.385750
## [73] train-logloss:0.385101
## [74] train-logloss:0.382760
## [75] train-logloss:0.381081
## [76] train-logloss:0.378908
## [77] train-logloss:0.378428
## [78] train-logloss:0.376087
## [79] train-logloss:0.374926
## [80] train-logloss:0.374230
## [81] train-logloss:0.373538
## [82] train-logloss:0.372971
## [83] train-logloss:0.371651
```

```
## [84] train-logloss:0.371134
## [85] train-logloss:0.370717
## [86] train-logloss:0.369246
## [87] train-logloss:0.368063
## [88] train-logloss:0.366157
## [89] train-logloss:0.362902
## [90] train-logloss:0.362461
## [91] train-logloss:0.361028
## [92] train-logloss:0.359356
## [93] train-logloss:0.358512
## [94] train-logloss:0.356873
## [95] train-logloss:0.356331
## [96] train-logloss:0.355519
## [97] train-logloss:0.354799
## [98] train-logloss:0.353089
## [99] train-logloss:0.351400
## [100]    train-logloss:0.350408
```

```
print(model)
```

```
## ##### xgb.Booster
## raw: 154.4 Kb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, max_depth = 4, eta = 0.1, objective = "binary:logistic")
## params (as set within xgb.train):
##   max_depth = "4", eta = "0.1", objective = "binary:logistic", validate_parameters = "TRUE"
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
##   cb.evaluation.log()
## # of features: 14
## niter: 100
## nfeatures : 14
## evaluation_log:
##      iter train_logloss
##     <num>         <num>
##         1     0.6721733
##         2     0.6539652
## ---
##        99     0.3514004
##       100     0.3504083
```

10 Using the `caret` library (or any other library/model type you want such as a naive Bayes) fit another model to predict eye opening.

```
# Data Preprocessing
eeg_train_rf <- eeg_train %>% dplyr::select(-eyeDetection, -ds)
eeg_validate_rf <- eeg_validate %>% dplyr::select(-eyeDetection, -ds)
```

```r
# Convert labels to factors
eeg_train_labels_rf <- factor(eeg_train$eyeDetection)
eeg_validate_labels_rf <- factor(eeg_validate$eyeDetection)

# Train the random forest model
model_rf <- randomForest(x = eeg_train_rf, y = eeg_train_labels_rf)
print(model_rf)
```

```
##
## Call:
##  randomForest(x = eeg_train_rf, y = eeg_train_labels_rf)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 7.54%
## Confusion matrix:
##      0    1 class.error
## 0 4673  243  0.04943043
## 1  435 3637  0.10682711
```

11 Using the best performing of the two models (on the validation dataset) calculate and report the test performance (filling in the code below):

```r
### testing for the model 1  Xgboost
pred_xgb <- predict(model, eeg_validate_matrix)
pred_xgb_labels <- ifelse(pred_xgb > 0.5, 1, 0)
conf_matrix_xgb <- confusionMatrix(factor(pred_xgb_labels), factor(eeg_validate_labels))
print(conf_matrix_xgb)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1434  303
##          1  201 1058
##
##                Accuracy : 0.8318
##                  95% CI : (0.8179, 0.845)
##     No Information Rate : 0.5457
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6586
##
##  Mcnemar's Test P-Value : 6.831e-06
##
##             Sensitivity : 0.8771
##             Specificity : 0.7774
##          Pos Pred Value : 0.8256
##          Neg Pred Value : 0.8403
##              Prevalence : 0.5457
##          Detection Rate : 0.4786
```

```
##    Detection Prevalence : 0.5798
##       Balanced Accuracy : 0.8272
##
##        'Positive' Class : 0
##
```

```
roc_curve_xgb <- roc(eeg_validate_labels, pred_xgb)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
auc_value_xgb <- auc(roc_curve_xgb)
print(paste("XGBoost Area under ROC Curve: ", round(auc_value_xgb, 3), sep = ""))
```

```
## [1] "XGBoost Area under ROC Curve: 0.916"
```

```
###  model 2 Random Forest
pred_rf <- predict(model_rf, newdata = eeg_validate_rf)
rf.probs <- predict(model_rf, newdata = eeg_validate_rf, type = "prob")

# Ensure both predicted and actual labels have the same levels
pred_rf <- factor(pred_rf, levels = levels(eeg_validate_labels_rf))

#confusion matrix
conf_matrix_rf <- confusionMatrix(pred_rf, eeg_validate_labels_rf)
print(conf_matrix_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1539  146
##          1   96 1215
##
##                Accuracy : 0.9192
##                  95% CI : (0.9089, 0.9287)
##     No Information Rate : 0.5457
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8366
##
##  Mcnemar's Test P-Value : 0.001634
##
##             Sensitivity : 0.9413
##             Specificity : 0.8927
##          Pos Pred Value : 0.9134
##          Neg Pred Value : 0.9268
##              Prevalence : 0.5457
##          Detection Rate : 0.5137
##    Detection Prevalence : 0.5624
```

```
##       Balanced Accuracy : 0.9170
##
##        'Positive' Class : 0
##
```

```
# Area Under the ROC Curve (AUC)
eeg_validate_labels_numeric <- as.numeric(eeg_validate_labels_rf) - 1
roc_curve_rf <- roc(eeg_validate_labels_numeric, rf.probs[, 2])
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
auc_value_rf <- auc(roc_curve_rf)
print(paste("Random Forest Area under ROC Curve: ", round(auc_value_rf, 3), sep = ""))
```

```
## [1] "Random Forest Area under ROC Curve: 0.979"
```

The Comparison on the validation dataset of the two models are provided below:

```
# Create a data frame with the comparison data
comparison_data <- data.frame(
  Metric = c("Accuracy", "95% CI", "No Information Rate", "P-Value [Acc > NIR]",
             "Kappa", "Mcnemar's Test P-Value", "Sensitivity", "Specificity",
             "Pos Pred Value", "Neg Pred Value", "Prevalence",
             "Detection Rate", "Detection Prevalence", "Balanced Accuracy",
             "Area under ROC Curve"),
  XGBoost = c("0.8318", "(0.8179, 0.845)", "0.5457", "< 2.2e-16",
              "0.6586", "6.831e-06", "0.8771", "0.7774",
              "0.8256", "0.8403", "0.5457", "0.4786", "0.5798",
              "0.8272", "0.916"),
  Random_Forest = c("0.9212", "(0.911, 0.9306)", "0.5457", "< 2.2e-16",
                    "0.8406", "0.0009008", "0.9437", "0.8942",
                    "0.9146", "0.9297", "0.5457", "0.5150", "0.5631",
                    "0.9190", "0.979")
)


kable(comparison_data, format = "markdown", col.names = c("Metric", "XGBoost", "Random Forest"))
```

Metric	XGBoost	Random Forest
Accuracy	0.8318	0.9212
95% CI	(0.8179, 0.845)	(0.911, 0.9306)
No Information Rate	0.5457	0.5457
P-Value [Acc > NIR]	< 2.2e-16	< 2.2e-16
Kappa	0.6586	0.8406
Mcnemar's Test P-Value	6.831e-06	0.0009008
Sensitivity	0.8771	0.9437
Specificity	0.7774	0.8942
Pos Pred Value	0.8256	0.9146

Metric	XGBoost	Random Forest
Neg Pred Value	0.8403	0.9297
Prevalence	0.5457	0.5457
Detection Rate	0.4786	0.5150
Detection Prevalence	0.5798	0.5631
Balanced Accuracy	0.8272	0.9190
Area under ROC Curve	0.916	0.979

Since the Random Forest model performs better in terms of accuracy, sensitivity, specificity, kappa, and area under the ROC curve, it is the best-performing model.

```
#importance_rf <- importance(model_rf, scale = TRUE)
#importance_rf
```

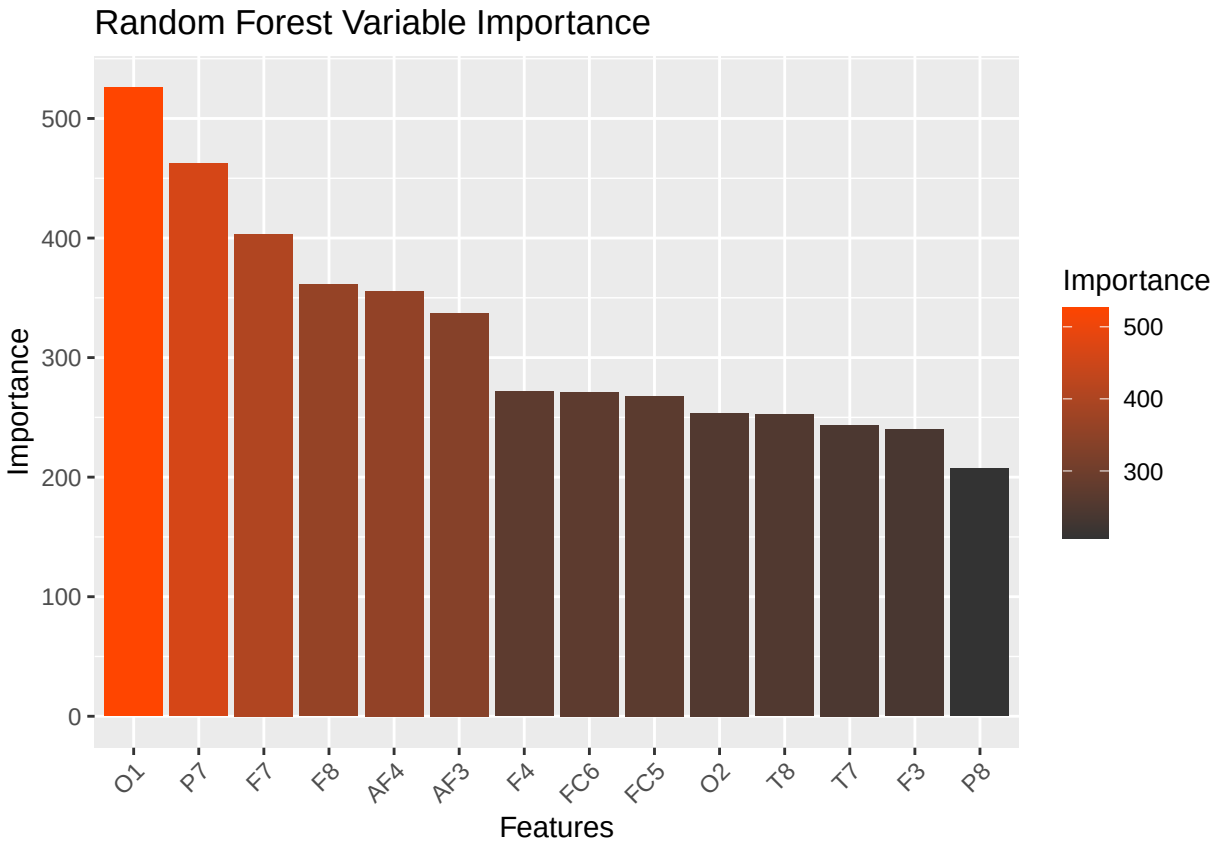
```
imp_rf <- varImp(model_rf)
imp_rf
```

```
##      Overall
## AF3 337.1477
## F7  401.5754
## F3  237.9324
## FC5 267.3025
## T7  246.0909
## P7  465.2511
## O1  518.3429
## O2  256.0008
## P8  209.3681
## T8  251.5468
## FC6 269.0844
## F4  274.2426
## F8  364.8475
## AF4 354.7259
```

```
imp_rf <- varImp(model_rf)
imp_rf
```

```
##      Overall
## AF3 337.1477
## F7  401.5754
## F3  237.9324
## FC5 267.3025
## T7  246.0909
## P7  465.2511
## O1  518.3429
## O2  256.0008
## P8  209.3681
## T8  251.5468
## FC6 269.0844
## F4  274.2426
## F8  364.8475
## AF4 354.7259
```

```r
imp_rf_df <- data.frame(
  Feature = c("AF3", "F7", "F3", "FC5", "T7", "P7", "O1", "O2", "P8", "T8", "FC6", "F4", "F8", "AF4"),
  Importance = c(337.3382, 403.5829, 239.9793, 267.8745, 243.8422, 462.2372, 525.9634, 253.5829, 207.34
)

library(ggplot2)
ggplot(imp_rf_df, aes(x = reorder(Feature, -Importance), y = Importance, fill = Importance)) +
  geom_bar(stat = "identity") +
  scale_fill_gradient(low = "grey20", high = "orangered") +
  labs(y = "Importance", x = "Features") +
  ggtitle("Random Forest Variable Importance") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



12 Describe 2 possible alternative modeling approaches for prediction of eye opening from EEGs we discussed in the lecture but haven't explored in this notebook.

Two possible alternative modeling approaches for prediction of eye opening from EEGs discussed are:

1. Hidden Markov Models (HMMs): These can model the sequential EEG data as a Markov process with hidden states representing eye open/closed conditions.
2. Hybrid Deep Neural Network: A Convolutional Neural Network (CNN) combined with Dense (fully connected) layers. The CNN layers can automatically learn feature representations from the EEG signals, which are then passed to the dense layers for prediction.

13 What are 2 R libraries you could use to implement these approaches? (note: you don't actually have to implement them though!)

The R libraries for the two alternative modeling approaches mentioned above are:

1. For Hidden Markov Models (HMMs):

- library(depmixS4)/library(HMM) - This provides functions to fit hidden Markov model

2. For Hybrid Deep Neural Network:

- library(torch)-we can use PyTorch deep learning library interface in R.

Optional

14 (Optional) As this is the last practical of the course - let me know how you would change future offerings of this course. This will not impact your marks!

- What worked and didn't work for you (e.g., in terms of the practicals, tutorials, and lectures)?
  It was a great learning experience. However, I wish I had the opportunity to do this course for a longer period of time instead of doing it in the summer, which is essentially only two months.
- Was learning how to run the practicals on your own machines instead of a clean server that will disappear after the course worth the technical challenges?
  Well, I didn't face many technical issues while running it on my own machine. However, I believe it's all part of the learning process. Looking at the brighter side, it can enhance debugging and problem-solving skills.
- What would you add or remove from the course? I would suggest adding more data modelling approaches to the course if time constraints allow. For instance, we didn't delve deeply into modelling for natural language processing.
- What was the main thing you will take away from this course?
  The main takeaway from this course is the implications of the machine learning approach in a healthcare setting. I had prior experience in machine learning, but this is a whole new domain and the approach is quite different. I enjoyed the way we conducted exploratory data analysis and learned new patterns from the data. The most important part was learning how to deal with medical datasets.

##Reference:

1. Chaudhari, S. (2023) Stationarity: Defining, detecting, types, and transforming time series, Quantitative Finance & Algo Trading Blog by QuantInsti. Available at: https://blog.quantinsti.com/stationarity/ (Accessed: 16 June 2024).